(2)

Towards Intelligent Automated Forces For Simnet

Semi-Annual Report to the Office of Naval Research
for the Period 10/1/91 - 3/31/92
on Contract Number N00014-91-J-1624

Paul S. Rosenbloom, PI
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

June 5, 1992

The research supported by this contract is focused on an initial investigation of the use of the Soar architecture in constructing autonomous intelligent agents that can act as independent players in multi-agent simulation environments, such as Simnet. The work is being pursued in the context of a simplified battlefield environment called *GridWorld*, which was originally developed at the Hughes AI Center, and then enhanced by us during the initial six months of this contract. GridWorld provides a two-dimensional, multi-agent, real-time world in which both space and time are represented as continuous quantities.

During the period covered by this report, we concentrated on two research activities: (1) investigating the integration of cognitive and spatial reasoning in a range of safe-traversal scenarios; and (2) extending GridWorld to allow multiple Soar-based agents to interact. The following sections provide additional detail on these two activities.

Integrating Cognitive and Spatial Reasoning

One of the greatest strengths of an integrated AI architecture, such as Soar, is that it should be able to facilitate the construction of agents that combine a wide range of knowledge and reasoning methods. On the other hand, one of Soar's greatest weaknesses is its lack of any demonstrable ability to use particular types of knowledge and reasoning, such as spatial knowledge and reasoning, that are critical to effective agent performance in worlds such as Simnet and GridWorld (or the real world, for that matter). Our goal for this segment of the effort was therefore to understand how spatial reasoning should be done in Soar for the GridWorld.

Our focus here is not on inventing new more optimal algorithms for, for example, navigation. There is already a strong research community focused on this, and it is not clear that Soar provides us with much direct leverage there (except, perhaps, in terms of building highly flexible knowledge-based navigational methods). Instead, we are focusing on the interactions between various forms of spatial reasoning and the rest of the cognitive system. This is a topic that has been little studied so far, and also one in which Soar can potentially provide great leverage.

To drive this effort we selected a class of safe-traversal tasks which stress a combination of spatial and non-spatial cognition. The tasks all involve two agents – one friendly agent (referred to as *R2*)and one hostile agent – in a two-dimensional world that include walls which block both movement and vision (Figure 1). R2's task is to travel between two points in the world without being caught. The hostile agent's task is to catch R2. Due to current limitations on the ability to simultaneously deploy multiple Soar-based agents in the GridWorld (but see the next section),

only R2 is Soar based. The hostile agent is a simple stateless Lisp-based agent that wanders around aimlessly[1] until it sees R2, chases it until it catches it or can no longer see it, and wanders aimlessly again if it can no longer see it.

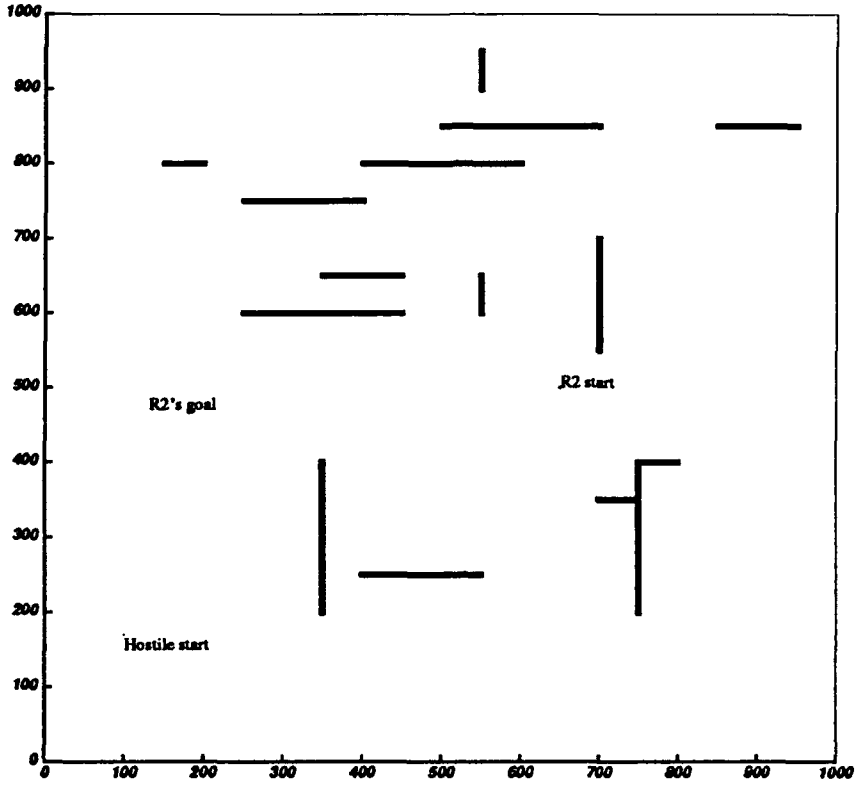

Figure 1: A GridWorld configuration for the safe-traversal task.

R2 has been the focus of our development efforts during the period covered by this report. An example trace of its current behavior can be seen in Figure 2. In this figure, R2 initially follows a plan based on criteria of safety and length. Once it perceives the hostile agent, it reacts immediately by starting to run away, while simultaneously selecting a hiding place (behind a wall) and generating a plan by which to get there. Once safely in its hiding place, it replans a route to its goal – possibly making use of rules learned during the original planning process – and once more proceeds on its way.[2]

In general, R2 begins by planning a route from its initial position to its goal position (Figure 3). It does this with the aid of a map in which the walls have been used to guide the decomposition of the world into rectangular regions of free space. The search for a path takes these regions as primitive nodes, and heuristically looks for a short, safe path. Shortness is determined by the obvious path-length metric. Safety is determined by a set of heuristics about possible ambush points; that is, about ends of walls behind which R2 can't see. The search

[1]It follows a straight line until it nears a wall or an edge of a world, in which case it makes a random turn and starts again in a straight line

[2]The odd looking jog in the replanned path at around (200, 700) occurs because the safety criteria dictates avoiding proximity to the ends of walls behind which it can't see. Safety is discussed in more detail below.

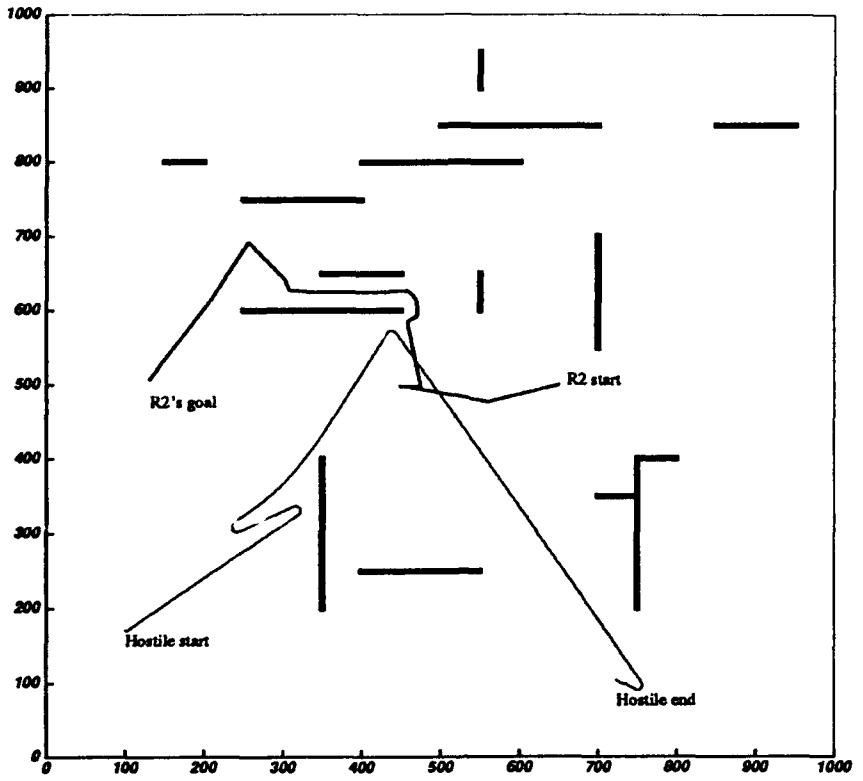

Figure 2: Behavioral trace of interactions between R2 and the hostile agent.

proceeds depth first, using a greedy heuristic based on distance to the goal to order the choices at each point, and backtracking whenever either an unsafe node or a dead-end is reached.[3] In the process, Soar's learning mechanism stores away new rules that capture the results of searching this tree of paths.

It then follows this path until the hostile agent is detected. Immediately, a reactive behavior is triggered off by a production sensitive to this detection. This behavior causes R2 to move away from the hostile agent. While running away, R2 uses a set of heuristics based on the relative location of the hostile agent and the near-by walls to select some hiding places that it hopes to be able to reach before it is caught (recall that the hostile agent gives up if it cannot see R2). It then plans a path to one of the hiding points, but using a very different strategy than during its initial path planning. Here it is important that the planning proceed quickly, and that the plan get the agent to its hiding place quickly. Issues of safety – at least with respect to possible ambush points – no longer matter, while the ability to take short-cuts matters a great deal. Furthermore, these short-cuts need to be sensible; i.e., they should not reduce the distance between R2 and the hostile agent. Given these constraints, especially the need for a quick response, the hiding planner relies on pre-computed (or ready-made) abstract plans. These abstract plans are instantiated for all of the potential hiding positions. Some of these plans are immediately rejected because their preconditions – which test for locations of different walls – are violated.

[3]In one set of systematic experiments with this path planner, out of 370 scenarios attempted, 300 were planned successful. Of the remaining 70 scenarios, 50 were terminated because of an imposed time-limit, and the remaining 20 failed because of a bug that has since been fixed.

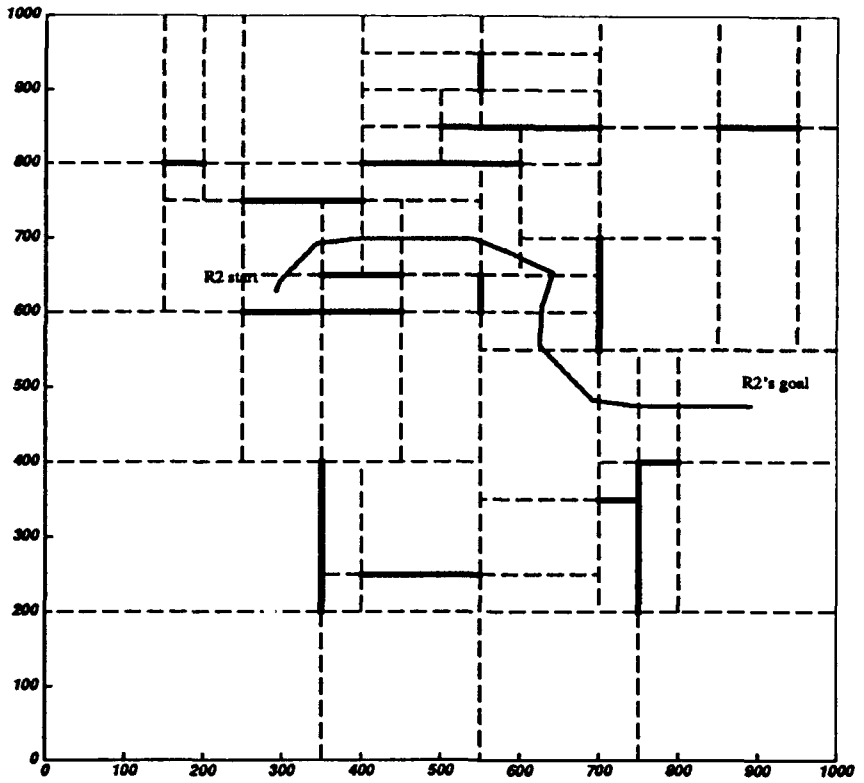

Figure 3: A short, safe path through free space.

The remaining plans are then heuristically ranked, and the best one is selected.

Figure 4 depicts two of the abstract plans used for hiding. The arrows represent individual steps in a plan, while the bold line represents a wall. As shown in the figure, the abstract plans usually have a length of two to three steps.

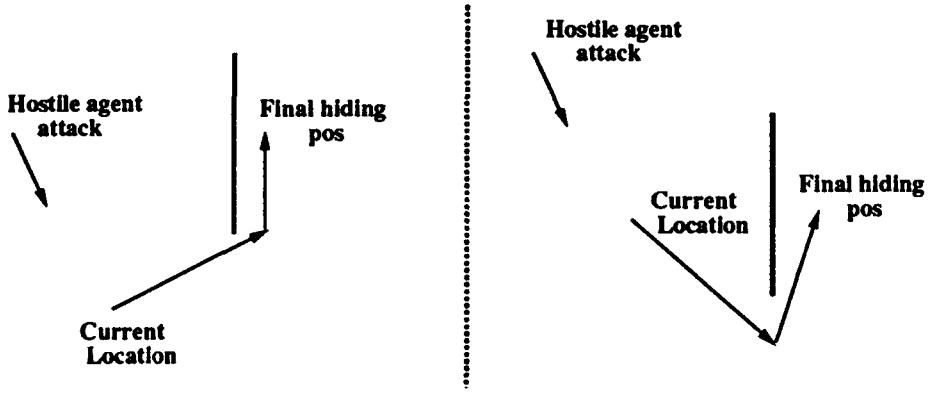

Figure 4: Two abstract hiding plans.

Each of the abstract plans have failure conditions associated with them. Based on simple spatial reasoning, the failure conditions test if a plan is doomed to fail. If so, that plan is abandoned. The planner goes back to its set of abstract plans to create and execute a new plan.

Note that the abstract plans allow R2 to move very close to the wall. This is unlike the plans

generated by the initial planner, where the path moves through centers of regions to avoid possible ambush points. Moving close to the walls allows R2 to take short cuts to the hiding spot. But these short-cuts extract a price – the uncertainty in the movement of R2 can cause it to bump into nearby walls. The problem becomes particularly severe if the walls have small uncertainties in their lengths (as they do in some of our experiments.) To avoid this, R2 has a built-in reactive wall-avoidance procedure, based on a *potential field.* A field is set up in which the obstacles repel R2, thus allowing R2 to avoid bumping into walls. It is this reactive behavior that causes the curved motion in R2's path execution. Note that in general, the potential-fields approach is problematical due to the presence of local minima. However, by using it in conjunction with the hiding planner, the problem of local minima is avoided.

Once the agent is safely hidden, it replans a path to its goal. (In many cases a good deal of what is learned during the initial planning process can transfer to this replanning process.) If the agent can now get safely to its goal, it will. However, if it once again meets up with the hostile agent, it will again try to hide and then replan once more.

When considered as a whole, R2 demonstrates a close integration of reaction, planning, execution, interruption, replanning, learning, search, use of heuristic knowledge, navigation, and abstract spatial reasoning (about hiding places). In many ways the behavior is still rather simple and special purpose – partially reflecting the fact that the opponent is itself quite simple – but it does provide a broad system that can form the basis for increased sophistication. Some weaknesses in the current design include R2's need for a reasonably accurate map (it cannot survive if no map is available), its inability to learn the map, its lack of any concept of time, its inability to form and use high-level strategies (other than its one current strategy of hiding), and an incomplete integration of vision and planning.

Extending GridWorld to Multiple Soar Agents

The GridWorld environment and all of the agents that interact within it currently run within a single Lisp image inside of a single Unix process.[4] This has been adequate for running one Soar-based agent amongst one or more Lisp-based agents, but not for running multiple Soar-based agents – Soar is currently structured under the assumption that there would be only one instantiation of it per Lisp image. Since the deployment of multiple Soar-based agents is critical to our ability to study interactions – of either an antagonistic or cooperative nature – among multiple intelligent agents, this became a high priority infrastructure problem to resolve.

Resolving this problem required making two basic design decisions: (1) whether the interactions should be synchronous or asynchronous; and (2) how to balance flexibility and cost. On synchrony, realistic environments – such as Simnet – clearly require asynchrony, but synchrony makes experimental repeatability much easier to achieve. We therefore provided both asynchronous and synchronous modes of operation. In the synchronous mode all agents have a fixed time to "think" and all the changes to the world occur within one simulation step. In the asynchronous mode, each agent thinks on its own and polls the environment and sends commands to it at its own rate.

---

[4]The original implementation from Hughes could be distributed over a set of Apple Macintoshes, but was not set up for Unix systems.

On implementation, there are a variety of approaches that could be taken, including: (1) repackaging Soar; (2) using a multi-process facility available within one of the versions of Lisp; (3) using multiple Unix processes communicating via pipes; (4) using multiple machines communicating via sockets; (5) using multiple processes on one or more machines communicating via network file systems. After reviewing these options, we chose (5) because of its ratio of cost to benefit. This option provides significant flexibility by transparently allowing the environment and agents to be within a single process, within different processes on a single machine, within processes on different machines on a local area network, or within different processes on different machines on a wide area network. The key to providing this flexibility is the availability of network file systems such as NFS and AFS that allow the local disk, a disk on a local network, or a disk elsewhere on the internet to be accessed in a uniform fashion. This option also leads to a relatively low-cost solution. It requires no modifications of Soar (as opposed to option (1)), and overall required a small amount of implementation effort.

The central idea behind the implementation is to use the file system as a simple message passing mechanism, in order to allow the agents to run in different processes. Each agent sees the file system as its own circular message buffer (mailbox). Messages are simply small files with a standardized name. Each agent keeps a private table of messages sent to and received from the rest of the agents, so it knows what messages to expect (filenames to look for) and to send (filenames to create). Messages contain ready-to-evaluate lisp expressions or some other data to be handled appropriately. The simulator is viewed as another agent acting as a server of world states. At every step of the simulation, it makes public the state of the environment in a shared file that is read by the rest of the agents. Each agent sends to the simulator commands describing the actions to be performed in the world. Each message contains the command lisp expression that is ready to evaluate by the simulator.

The Next Steps

During the next period we expect to focus on three key topics. The first topic is generalizing the agent's spatial/navigational abilities to provide a more sophisticated and flexible facility. This includes some or all of examining strategies for generalizing and integrating together R2's two current planners, adding the ability to work flexibly with multiple planning criteria (such as planning time, execution time, and safety) and multiple types of knowledge (such as partial or complete map knowledge, verbal directions, and partial or complete models of the hostile agent's location and/or behavior patterns), and the integration and use of external tools (such as A* search programs and visibility-graph generators) to augment its internal capabilities. The second topic is the construction of a Soar-based hostile agent. This will provide its own set of challenges, as well as forcing R2 to increased levels of sophistication. The third topic is the combination of multiple Soar-based versions of R2. This should raise issues of cooperation and communication that do not exist in the single agent (or even one-on-one) situation.